Attorney Docket No. 42P8534C

PATENT

UNITED STATES PATENT APPLICATION

FOR

SYMBOLIC MODEL CHECKING WITH DYNAMIC MODEL PRUNING

INVENTORS:

JIN YANG

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8598

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EV336586078US

Date of Deposit September 17, 2003

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Alexandria, VA 22313-1450

Anne Collette
(Typed or printed name of person mailing paper or fee)

Anne Collette 9/17/2003
(Signature of person mailing paper or fee) Date

SYMBOLIC MODEL CHECKING WITH DYNAMIC MODEL PRUNING

RELATED APPLICATIONS

This is a continuation of application Ser. No. 09/677,262 filed September 30, 2000, which is currently pending.

FIELD OF THE INVENTION

This invention relates generally to automated design verification, and in particular to more efficient use of binary decision diagrams to perform automated symbolic model checking for very large scale integrated circuit designs and other finite state systems.

BACKGROUND OF THE INVENTION

Modern design of very large-scale integrated circuits often involves years of research and the efforts of hundreds of engineers. Automated formal verification methods are an essential part of the design effort, reducing errors, lost time and risk to financial investment. Formal verification involves building a finite model of a system as a set of states and state transitions and checking that a desired property holds in the model. An exhaustive search of all possible states of the model may be performed in order to verify a desired property.

As the size and complexity of designs increase, much effort is expended to improve the efficiency of automated formal verification methods. One technique used in symbolic model checking to improve efficiency is to employ binary

decision diagrams (BDDs). A BDD is a directed acyclic graph that represents a Boolean expression. For each Boolean variable, there are two outgoing edges representing true or false assignments to the variable. The use of BDDs permits computation times, which are some polynomial function of the number of expression variables. Alternative representations such as clauses or truth tables require execution times, which are some exponential function of the number of expression variables. Therefore, use of BDDs has been popular in the formal verification community since the late 1980's.

BDDs, however, are not without drawbacks. The ordering of variables is critical to an efficient use of BDDs. Poor variable ordering can increase a BDDs size and cause exponential execution times. One method for symbolic model checking using BDDs comes from Carnegie Mellon University and is known as Symbolic Model Verifier (SMV).

Alternatively SMV uses a well known heuristic based procedure named *simplify_assuming* that is aimed at reducing BDD representations by simplifying a predicate using an invariant assumption but introduces a proof obligation, which must also be verified. Since the assumption is static it may also be ineffective in pruning a model.

Over the years, techniques have been developed to improve performance and capacity of BDD-based algorithms. One technique is called Cone of Influence (COI) reduction. In COI reduction, an abstraction is built for a circuit model consisting of next state functions only for variables in the dependency closure of variables of interest in the circuit specification. One drawback is that

all variables in the dependency closure do not necessarily influence the variables of interest in the circuit specification. A second drawback is that the abstraction that is built and used for each model-checking step may include portions that are useful in only a few of the model checking steps. Therefore needless extra computations are potentially performed, resulting in little benefit to the circuit verification.

Some methods have attempted to improve upon COI reduction by starting from a small portion of the dependency closure and extending the portion only when model checking fails to produce a satisfactory result. But these techniques also perform unnecessary computations on portions that are not relevant to the particular model-checking step being performed.

One method called the bounded cone of influence (BCOI) was proposed by A. Biere et al for symbolic model checking without BDDs [A. Biere, E. Clark, R. Raimi, and Y. Zhu; Verifying safety properties of a PowerPC™ microprocessor using symbolic model checking without BDDs; CAV'99; 1999]. However, even the BCOI method potentially includes irrelevant variables in the abstraction it builds, and the technique is not applicable to improve the widely used BDD-based approaches.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings.

Figure 1 illustrates an example of a circuit.

Figure 2a graphically illustrates a transition relation for a circuit.

Figure 2b shows another transition relation built as part of a lazy pre-image computation.

Figure 3a illustrates one embodiment of a method for performing lazy pre-image computations.

Figure 3b illustrates one embodiment of a more detailed method for performing lazy pre-image computations.

Figure 4a illustrates one embodiment of a method for computing a fixpoint using lazy pre-image computations.

Figure 4b shows an example of a lazy fixpoint computation for a circuit.

Figure 5a illustrates a parsing of a property to be evaluated.

Figure 5b. illustrates one embodiment of a method for producing and propagating assumptions from sub-properties to be evaluated.

Figure 5c. shows an example of producing and propagating assumptions from sub-properties to be evaluated.

Figure 6a graphically illustrates the state space of a model with its transition relation built using lazy pre-image computations to evaluate sub-property 530.

Figur 6b graphically illustrates the state space of a dynamically pruned model with its transition relation built using lazy pre-image computations to evaluate sub-property 531 under assumption 541.

Figure 6c graphically illustrates the state space of a dynamically pruned model with its transition relation built using lazy pre-image computations to evaluate sub-property 532 under assumption 542.

Figure 6d graphically illustrates the state space of a dynamically pruned model with its transition relation built using lazy pre-image computations to evaluate sub-property 533 under assumption 543.

Figure 7a illustrates one embodiment of a method for dynamically pruning a model by producing and propagating assumptions from sub-properties to be evaluated and pruning the transition relation under the assumptions generated.

Figure 7b an example of dynamically pruning a transition relation for a model under assumptions generated from sub-properties to be evaluated.

Figure 8 depicts a computing system for automated lazy symbolic model checking of finite state systems using symbolic variable reduction.

DETAILED DESCRIPTION

Embodiments of the present invention may be realized in accordance with the following teachings and it should be evident that various modifications and changes may be made in the following teachings without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense and the invention measured only in terms of the claims.

Methods for formal verification of circuits and other finite-state systems are disclosed herein providing improved efficiency and capacity of popular binary decision diagram (BDD) based algorithms. For one embodiment of a lazy pre-image computation, a method is disclosed that builds new transition relation partitions on-demand only for relevant next internal variables of a state predicate, and conjoins only next state relations for relevant next internal variables to a pre-image including the state predicate. For one embodiment of a lazy fixpoint computation, a method is disclosed that makes iterative use of lazy pre-image computation to compute conditions that must necessarily be satisfied to produce a given set of states. For one embodiment of forward assumption propagation, a method is disclosed that generates assumptions to characterize a set of interesting states for sub-properties of the property being evaluated at one or more evaluation stages. For one embodiment of a dynamic transition relation pruning technique, a method is disclosed that improves the efficiency for symbolic model checking computations by pruning transition relations under assumptions dynamically generated from the properties being evaluated, thereby

providing means to handle very large scale integrated circuits and other finite state systems of problematic complexity for prior methods. The teachings of these disclosed methods provide for symbolic model checking of circuits and other finite state systems previously too large to be completed successfully using BDD based algorithms.

Figure 1 illustrates an example of a circuit 101 having internal state variables c, d, e and f; and input variables a and b. According to the logical combination of inputs to memory element 102, the value of internal state variable c at its next transition can be determined to be the value of the Boolean expression a AND c. From the logical combination of inputs to memory element 103, the value of internal state variable d at its next transition can be determined to be the value of the input variable b. From the logical combination of inputs to memory element 104, the value of internal state variable e at its next transition can be determined to be the value of the Boolean expression c OR e. Finally, from the logical combination of inputs to memory element 105, the value of internal state variable f at its next transition can be determined to be the value of the Boolean expression c NAND d.

A model of a circuit or other finite state system can be formally defined as: a nonempty finite set of Boolean variables, $V=V_S \cup V_I$, consisting of a union V of internal state variables V_S with input variables V_I; and a next state function N(v) for each v in V_S, which is an assignment mapping of internal state variables according to Boolean (true or false) valued expressions on V called state predicates.

For one embodiment, a model may be represented as a nonempty transition relation, R, on state predicate pairs, where (S1, S2) is an element of the transition relation, R, if there exists a transition in the finite state system from a state satisfying predicate S1 to state satisfying predicate S2. R is also referred to as a model of the finite state system.

A partitioned transition relation, R, on a partitioning of the internal state variables {V1, V2, ... , Vk} has the implicitly conjoined form:

$$R(V, V') = R1(V,V1') \text{ AND } R2(V,V2') \text{:... AND } Rk(V,Vk')$$

where the ith partition is $Ri(V,Vi') = AND_{\text{for all v' in Vi'}} (v'=N(v))$. The assertion v'=N(v) is called the next state relation for v and v' is a copy of v to record the value taken on by v at the next transition.

The set of states, S, may be represented using a Boolean state predicate S(V). Operations on sets may be carried out as algebraic manipulations of state predicates. The set of states that can move to S in one transition is called the pre-image of S and written

$$Pre(S(V)) = \exists V'.[\ AND_{\text{for all v' in Vs'}} (v'=N(v)) \text{ AND } S(V')].$$

An existential operation $\exists V'.[S(V')]$ represents a quantification of state predicate S(V') over the variables in V'. Typically, in order to more efficiently use computation resources, the operation of computing the pre-image of a set of states is carried out as a relation product of state predicates using early variable quantification for partitioned transition relations, thereby permitting staged reductions of Boolean expressions, as follows:

$$Pre(S(V)) = \exists V1'.[R1(V,V1') \text{ AND } ($$

$\exists V2'.[R2(V, V2')$ AND (

...

$\exists Vk.[Rk(V,Vk')$ AND (

$\exists Vl'.S(V')$)]

...)]

)].

An alternative definition for a model can be set forth as a pair of induced transformers, Pre and Post, such that Pre(S2) includes S1 and Post(S1) includes S2 if (S1,S2) is an element of R. In other words, the Pre transformer identifies any states satisfying predicate S, for which there exists a transition to some state satisfying predicate S'. Pre is called a pre-image transformer. The Post transformer identifies any states satisfying predicate S', for which there exists a transition from some state satisfying predicate S. Post is called a post-image transformer.

One drawback of a typical pre-image computation is that it involves the entire partitioned transition relation. But S(V) may involve only a few variables. Consequently, not all next state relations are relevant in any particular invocation of a pre-image computation.

For example, Figure 2a graphically illustrates a possible transition relation 201 for circuit 101 having V_S={c, d, e, f} and V_I={a, b}. The next state function for variable c is N(c)=a AND c. Therefore, in order for the circuit to reach a state, S(c), where c=1 it must have made transition 211 from a state S(a AND c) where

a=1 and c=1. The next state function for variable d is N(d)=b. Therefore, in order for the circuit to reach a state, S(d), where d=1 it must have made transition 219 from a state S(b) where b=1. The next state function for variable e is N(e)=e OR c. Therefore, in order for the circuit to reach a state, S(e), where e=1 it must have made transition 212 from a state S(c) where c=1 or it must have made transition 213 from a state S(e) where e=1. The next state function for variable f is N(f)=d NAND c. Therefore, in order for the circuit to reach a state, S(f), where f=1 it must have made transition 215 from a state S(NOT c) where c=0 or it must have made transition 218 from a state S(NOT d) where d=0.

Computing all states reachable to S(e) in two or more transitions includes the next state function for variable c, which has already been shown as N(c)=a AND c represented by transition 211. The next state function for variable NOT c is N(NOT c)=NOT(a AND c)=(NOT a) OR (NOT c). Therefore, in order for the circuit 101 to reach a state, S(NOT c), where c=0 it must have made transition 214 from a state S(NOT a) where a=0 or it must have made transition 216 from a state S(NOT c) where c=0. The next state function for variable NOT d is N(NOT d)=NOT b. Therefore, in order for the circuit to reach a state, S(NOT d), where d=0 it must have made transition 217 from a state S(NOT b) where b=0.

For a given state predicate, an invocation of a pre-image computation that uses transition relation 201 may result in computations that are not relevant to that state predicate. For one embodiment, a lazy pre-image computation is disclosed which provides a relevant transition relation abstraction for each pre-

image computation according to the state predicate of the invocation. Such a lazy pre-image computation may be performed for a state predicate S(W), where W is contained in V and W_S' is the set of next internal variables in the set of next variables W', as follows:

$$\text{Pre}(S(W)) = \exists W'.[\ \text{AND}_{\text{for all v' in Ws'}}\ (v'=N(v))\ \text{AND}\ S(W')].$$

The approach provided by the lazy pre-image computation disclosed above differs from previous COI reduction approaches in that it is not statically derived from a model specification and then used throughout. Instead, it dynamically provides an abstraction for each pre-image computation that is relevant to the particular state predicate associated with the invocation. Accordingly, lazy pre-image computation provides for greater efficiency and capacity improvements in popular BDD-based symbolic model checking methods than previously used pre-image computation methods.

For example, the lazy pre-image of a state predicate S(e) for circuit 101 where e=1 can be computed:

$$\begin{aligned}\text{Pre}(S(e)) &= \exists e'.[(e'=N(e))\ \text{AND}\ S(e')].\\ &= \exists e'.[(e'=e\ \text{OR}\ c)\ \text{AND}\ e'].\\ &= (e\ \text{OR}\ c).\end{aligned}$$

Figure 2b graphically illustrates a possible transition relation 202 for circuit 101 built as a result of an invocation of the lazy pre-image computation Pre(S(e)) on the state predicate S(e) where e=1. The next state function for variable e is N(e)=e OR c. Therefore, in order for the circuit to reach a state, S(e), where e=1 it must have made transition 222 from a state S(c) where c=1 or it must have

made transition 223 from a state S(e) where e=1. Since no other transitions are relevant to reaching state S(e), the lazy pre-image method need not build them. As seen in the above example, this lazy pre-image method potentially reduces the number of transition relation partitions involved and also the sizes of partitions. Therefore computations required to explicitly build a BDD for a desired function may be significantly reduced.

For one embodiment, Figure 3a illustrates performing a lazy pre-image computation. In processing block 311 transition relation partitions are updated as needed by adding new transition relations for only the relevant next internal variables. In processing block 312 a pre-image is initialized to the next state predicate of the invocation and existentially quantified over the relevant next input variables. In processing block 313, partitions with relevant next variables are identified. Finally in processing block 314, next state relations for relevant variables from the partitions identified in processing block 313 are conjoined to the pre-image and quantified.

The lazy pre-image method disclosed above provides for greater efficiency and capacity for symbolic model checking operations, particularly on circuits with a large number of variables. In a BDD based implementation, building transition relation partitions only as needed and only for relevant next internal variables is especially beneficial since the next state function for an internal variable is efficiently and implicitly encoded, but a BDD for the function must be explicitly built for symbolic model checking. Explicitly building BDDs unnecessarily may become computationally expensive.

Figure 3b details one embodiment of a method for performing a lazy pre-image computation on a state predicate S(W) involving a set W of internal variables and input variables. In processing block 320, W_S' is initialized to be the set of next internal variables in W'. In processing block 321, W_I' is initialized to be the set of next input variables in W'. In processing block 322, the next internal variables are checked to identify some variable w' that has not been evaluated. If one is identified, w'=N(w) is conjoined to the partition including w' and flow returns to processing block 322 to look for more next variables that have not been evaluated. Thus the transition relation partitions are built as needed for the relevant next internal variables. When no more are found, flow proceeds at processing block 324. In processing block 324 the pre-image is initialized to the state predicate existentially quantified for the relevant next input variables and partition counter i is set to k+1. In processing block 325, i is decremented. Then in processing block 326, partition counter i is tested to see if it has reached zero. If partition counter i has not reached zero, in processing block 327 partition Vi' is checked against W' to identify relevant variables. If no relevant variables are found, partition Vi' is skipped and flow proceeds at processing block 325. Otherwise in processing block 328, all next variables in Vi' that are not in W' are existentially quantified out from partition Vi' and the remaining relevant variables are evaluated according to their next state relations and assigned to Ra. Then in processing block 329, Ra is conjoined with the pre-image Pre and flow proceeds with the next i at processing block 325. When i=0

indicating no more partitions remain at processing block 326, the processing terminates and pre-image Pre is complete.

In one embodiment, the lazy pre-image computation disclosed above provides for potential improvements in key model checking techniques. For example one embodiment of a lazy pre-image method provides an efficient general operation that may also be used effectively in performing fixpoint computations.

Figure 4a illustrates one embodiment of a fixpoint computation method which uses lazy pre-image computations. In processing block 411, a partial fixpoint state predicate, Fix_0, and an initial frontier predicate, $Front_0$, are both set to the input predicate S(W), and counter i is initialized to 1. In processing block 412, the new frontier predicate, $Front_i$, is set to the lazy pre-image of the previous frontier predicate, $Front_{i-1}$, intersected with the negated partial fixpoint predicate, $\neg Fix_{i-1}$, in order to exclude any states whose pre-images have already been computed. This computation is expressed symbolically as $Pre(Front_{i-1}) \wedge \neg Fix_{i-1}$. In processing block 413 a new fixpoint predicate Fix_i is set to the union of the new frontier predicate, $Front_i$, and the previous partial fixpoint predicate, Fix_{i-1}. Counter i is then incremented. In processing block 419, $Front_i$ is tested to see if any states from the previous iteration that need to have pre-images computed remain in the frontier. If so, processing beginning at processing block 412 repeats until $Front_i$ is emptied of such states, in which case processing terminates at processing block 419.

Figure 4b illustrates an example for one embodiment of performing a lazy fixpoint computation for state predicate, S(e), where e=1, on circuit 101. The fixpoint Fix_0 predicate 420 for the states reachable to S(e) in zero transitions and the frontier $Front_0$ are initially set to e. Since no pre-image computation is required, no transition relation is built. To compute the fixpoint Fix_1 predicate 421 for the states reachable to S(e) in one transition a lazy pre-image of the frontier predicate $Front_0$ is computed and combined with NOT Fix_0. Since frontier predicate $Front_0$ only involves signal e, lazy transition relation building only computes a transition relation partition for e, as [N(e)=e OR c]. Lazy pre-image Pre(S(e)) can be computed as previously shown, and the lazy pre-image computation returns e OR c based on the partially computed transition relation. The new frontier predicate $Front_1$ is set to (e OR c) AND NOT e in accordance with processing block 412, which reduces to c AND NOT e. Fixpoint Fix_1 predicate 421 for states reachable to S(e) in one transition is set to (c AND NOT e) OR e, which becomes e OR c.

To compute the fixpoint Fix_2 predicate 422 for those states reachable to S(e) in two transitions, the lazy pre-image of the frontier predicate $Front_1$ is computed and combined with NOT Fix_1. The pre-image is calculated as follows:

$$\text{Pre}(c \text{ AND NOT } e) = \exists e', c'.[(e'=N(e)) \text{ AND } (c'=N(c)) \text{ AND } S(e', c')].$$

$$= \exists e', c'.[(e'=e \text{ OR } c) \text{ AND } (c'=c \text{ AND } a) \text{ AND } (c' \text{ AND NOT } e')].$$

$$= (c \text{ AND } a) \text{ AND NOT } (e \text{ OR } c).$$

Predicate (c AND NOT e) requires lazy transition relation building of the translation relation partition for c, as [N(c)=c AND a]. Lazy pre-image

computation returns (c AND a) AND NOT (e OR c) based on the partially computed transition relation. The new frontier predicate $Front_2$ is set to (c AND a) AND NOT (e OR c) in accordance with processing block 412, which reduces to (c AND a AND NOT e AND NOT c) = 0. Fixpoint Fix_2 Predicate 422 for states reachable to S(e) in two transitions becomes just (e OR c).

Since frontier predicate $Front_2 = 0$ the lazy fixpoint computation terminates. The transition relations for b, d and f are not needed and therefore they are not built.

It will be appreciated that a symbolic model checking operation may benefit significantly from a reduction in the number of transition relations used to represent a model. A further reduction may be accomplished if for each property to be evaluated, only the necessary portions of the model are represented. For one embodiment of a dynamic transition relation pruning technique, a method is disclosed that improves the efficiency for symbolic model checking computations by pruning transition relations under assumptions dynamically generated from the properties being evaluated, thereby providing means to handle very large scale integrated circuits and other finite state systems of problematic complexity for prior methods.

Figure 5a illustrates a parsing of a property 510, $a \Rightarrow (b \Rightarrow X(Xf))$. At the first stage the property is parsed into a root 500 representing the logical implication operation, a left sub-property 506 representing the variable, a, and a right sub-property 511 representing $b \Rightarrow X(Xf)$. The operator X indicates that its predicate argument holds at the next transition.

At the second stage the sub-property 511 is parsed into a root 501 representing the logical implication operation, a left sub-property 507 representing the variable, b, and a right sub-property 512 representing X(Xf). At the third stage the property is parsed into a root 502 representing the next state operator X, and a right sub-property 513 representing Xf. Finally at the fourth stage the sub-property 513 is parsed into a root 503 representing the next state operator X, and a right sub-property 514 representing f. Given a parsing of the property assumptions may be generated for the sup-properties to be evaluated.

Figure 5b illustrates one embodiment of a method for producing and propagating assumptions from sub-properties to be evaluated. In processing block 521, the assumption for iteration zero, $Assum_0$, is initialized to the value one (true) and Node is set to the root 500 of the property to be evaluated. The iteration counter i is then incremented and processing proceeds to processing block 522. In processing block 522, the Node is tested to see if it consists of a single variable, in which case processing terminates at processing block 522. If not, processing proceeds to processing block 523. In processing block 523 the type of the Node operation is identified. If it is an implication operation processing proceeds at processing block 524. On the other hand, if it is a next state operator X, then processing proceeds to processing block 525.

In processing block 524 the assumption for iteration i, $Assum_i$, is set to the assumption for iteration i-1, $Assum_{i-1}$, combined with the left sub-property of Node using the logical AND operation. In processing block 525 the assumption for iteration i, $Assum_i$, is set to post-image of the assumption for iteration i-1,

$Assum_{i-1}$. Processing then proceeds to processing block 526 from processing block 524 or from processing block 525, where Node is set to the right sub-property of Node. The iteration counter i is then incremented and processing proceeds to processing block 522.

Figure 5c shows an example of producing and propagating assumptions from sub-properties to be evaluated. In iteration zero, assumption 540 is set to the value one and sub-property 530 is set to the state predicate for property to be evaluated ($a \Rightarrow (b \Rightarrow X(Xf))$). In iteration one, assumption 541 is set to a = (1 AND a) in accordance with processing block 524 and sub-property 531 is set to the right sub-property of sub-property 530, $b \Rightarrow X(Xf)$ in accordance with processing block 526. In iteration two, assumption 542 is set to a AND b and sub-property 532 is set to the right sub-property of sub-property 531, X(Xf). In iteration three, assumption 543 is set to Post(a AND b), which may be evaluated as d since N(d) = b, and sub-property 533 is set to the right sub-property of sub-property 532, Xf. In iteration four, assumption 544 is set to Post(Post(a AND b) which may be evaluated to one (true) and sub-property 534 is set to the right sub-property of sub-property 533, f.

The number of variables in a transition relation may be reduced according to a dynamically generated assumption as the transition relation is built. For instance the next state function for a variable f, N(f) = c NAND d may be pruned according to an assumption including (d=1) to N(f) = NOT c.

Figure 6a graphically illustrates the state space of a model with a transition relation that was built according to one embodiment of an iterative lazy pre-

image computation method to evaluate sub-property 530 or 534. It includes five variables that may be exhaustively searched for an assignment that satisfies the sub-property 530 or 534. These variables are f at block 611, c at block 613, d at block 615, a at block 617 and b at block 618. The three internal variables for which next state functions are included in the transition relation are, N(f) at block 612, N(c) at block 614, and N(d) at block 616. Since the assumptions 540 and 544 are trivial no pruning may be performed on the next state functions.

Figure 6b graphically illustrates the state space of a dynamically pruned model with a transition relation that was built according to one embodiment of a lazy pre-image computation method to evaluate sub-property 531 under assumption 541. It includes four variables that may be exhaustively searched for an assignment that satisfies the sub-properties 531 or 532. These variables are f at block 621, c at block 623, d at block 625, and b at block 628. The three internal variables for which next state functions are included in the transition relation are, N(f) at block 622, N(c) at block 624, and N(d) at block 626. Since the assumption 541 includes only the input variable, a, pruning of the transition relation may be performed only on the next state function for c, producing N(c)=c instead of N(c)=c AND a.

Figure 6c graphically illustrates the state space of a dynamically pruned model with a transition relation that was built according to one embodiment of a lazy pre-image computation method to evaluate sub-property 532 under assumption 542. It includes three variables that may be exhaustively searched for an assignment that satisfies the sub-property 531. These variables are f at

block 621, c at block 623, d at block 625. The three internal variables for which next state functions are included in the transition relation are, N(f) at block 622, N(c) at block 624, and N(d) at block 626. Since the assumption 541 includes the input variables, a and b, pruning of the transition relation may be performed on the next state function for c, producing N(c)=c instead of N(c)=c AND a and on the next state function for d, producing N(d)=1 instead of N(d)=b.

Figure 6d graphically illustrates the state space of a dynamically pruned model with a transition relation that was built according to one embodiment of a lazy pre-image computation method to evaluate sub-property 533 under assumption 543. It includes only two variables that may be exhaustively searched for an assignment that satisfies the sub-property 533. These variables are f at block 631 and c at block 633. Both of these internal variables' next state functions are included in the transition relation. They are, N(f) at block 632, and N(c) at block 634. Since the assumption 543 includes the input variable, a, and the internal variable, d, pruning of the transition relation may be performed on the next state functions for c, producing N(c)=c instead of N(c)=c AND a and for f, producing N(f)=NOT c instead of N(f)=c NAND d.

It will be appreciated that a property may be considered a sub-property of itself. It will also be appreciated that an assumption produced from a sub-property to be evaluated may be used to prune a transition relation in a variety of ways. It can be observed in Figure 6a through Figure 6e that a significant reduction in state storage may be achieved through dynamic model pruning.

Alternatively the computational complexity of checking the model may be reduced according to the assumption. For instance some model checking methods exhaustively search a state space in order to verify a property or sub-property. If the transition relation is pruned by one or more assignments of variables according to the assumption produced from a sub-property of the property being evaluated, the computational complexity of the search may be significantly reduced.

It will also be appreciated that one may desire to reduce the overall size of the model representation, perhaps at the cost of some addition computational complexity. Alternatively the number of variables in a model or the complexity of evaluating a sub-property may be reduced according to the assumption as the sub-property is being evaluated on the transition relation.

For example, a lazy pre-image of a state predicate S(Xf) AND $Assum_3$ for circuit 101 can be computed:

$$Pre(S(Xf)) \text{ AND } Assum_3 = \exists f'.[(f'=N(f)) \text{ AND } S(f')] \text{ AND } Assum_3$$

$$= \exists f'.[(f'=(c \text{ NAND } d)) \text{ AND } f'] \text{ AND } Assum_3 = (c \text{ NAND } d) \text{ AND } d=1$$

$$= \text{NOT } c$$

In general each sub-property may need to be evaluated according to its corresponding assumption. This operation may be performed in such a way as to substantially reduce the size of transition relations by pruning next state functions as the transition relations are dynamically built by a lazy pre-image calculation.

Figure 7a illustrates one embodiment of a method for dynamically pruning a model by producing and propagating assumptions as shown in Figure 5b from sub-properties to be evaluated and then pruning the transition relation under the assumptions generated. Processing block 711 is entered when a singular Node is identified in processing block 522. In processing block 711, the iteration counter i is decremented and then the sub-property for iteration i, SP_i is computed from the singular Node variable under the assumption for iteration i, $Assum_i$. Processing then proceeds to processing block 712 where the iteration counter is tested. Processing beginning in processing block 713 is repeated until the iteration counter is equal to zero in processing block 712, in which case processing terminates and returns the predicate SP_i. In processing block 713 the Node is set to its parent Node and the iteration counter, i, is decremented. Processing then proceeds in processing block 714.

In processing block 714 the type of the Node operation is identified. If it is an implication operation processing proceeds at processing block 715. On the other hand, if it is a next state operator X, then processing proceeds to processing block 716.

In processing block 715 the state predicate for iteration i, SP_i, is set to the state predicate for iteration i+1, SP_{i+1}, combined with the negated left sub-property of Node using the logical OR operation and evaluated under the assumption $Assum_i$. In processing block 716 the state predicate for iteration i, SP_i, is set to lazy pre-image of the state predicate for iteration i+1, SP_{i+1} and

evaluated under the assumption $Assum_i$. Processing then proceeds to processing block 712 from processing block 715 or from processing block 716.

Figure 7b shows an example of dynamically pruning a transition relation as it is built in a lazy pre-image computation according to assumptions generated from sub-properties to be evaluated. In iteration 4, state predicate 724 is set to the singular Node variable, f, and evaluated under the assumption 1 (true) which leaves the predicate unchanged. In iteration 3, state predicate 723 is set to the lazy pre-image of the predicate of the predicate f and evaluated under the assumption, d, which in accordance with processing block 716, reduces the next state function N(f)=c NAND d to N(f)=NOT c. In iteration 2, state predicate 722 is set to the lazy pre-image of the predicate, NOT c, and evaluated under the assumption, a AND b, which in accordance with processing block 716, reduces the next state function N(c)=a AND c to N(c)= c. In iteration 1, state predicate 721 is set to the logical OR combination of predicate, NOT c, and negated left sub-property, b, resulting in (NOT b OR NOT c) and then evaluated under the assumption, a, in accordance with processing block 715, which leaves the predicate unchanged. In iteration 0, state predicate 720 is set to the logical OR combination of predicate, (NOT b OR NOT c), and negated left sub-property, a, resulting in (NOT a OR NOT b OR NOT c) and then evaluated under the assumption, a, in accordance with processing block 715, which leaves the predicate unchanged.

In general, each sub-property may need to evaluated according to its corresponding assumption. If the lazy pre-image computation $Pre(SP_{i+1})$ under

the assumption $Assum_i$ is equal to 1 then no succeeding evaluations are needed as each will trivially be satisfied in processing block 715 or processing block 716.

It will be appreciated that the methods herein disclosed or methods substantially similar to those herein disclosed may be implemented in one of many programming languages for performing automated computations including but not limited to lazy pre-image computations, dynamic production and propagation of assumptions, dynamic pruning of transition relations, lazy fixpoint computations using dynamic model pruning, and lazy model checking using dynamic model pruning on high-speed computing devices.

For example, Figure 8 illustrates a computer system to perform computations, for one such embodiment. Processing device 822 is connectable with various recordable storage media, transmission media and I/O devices to receive data structures and programmed methods. Representative data structures 801 may include circuit descriptions 811, transition relations 812, and finite state models 813. Representative programmed methods 802 may include assumption propagation programs 814, lazy pre-image programs 815, transition relation pruning programs 816, and model checking programs 817. Components of either or both of the data structures and programmed methods may be stored or transmitted on recordable media such as removable storage disks 825, which may be accessed through an access device 826 in processing device 822 or in a storage serving system 821. Storage serving system 821 or processing device 822 may also include other removable storage media or non-removable storage media suitable for storing or transmitting data structures 801 or programmed

methods 802. Component data structures and programmed methods may also be stored or transmitted on transmission media such as network 824 for access by processing device 822 or entered by users through I/O device 823. It will be appreciated that systems such as the one illustrated are commonly available and widely used in the art of designing finite state hardware and software systems. It will also be appreciated that the complexity, capabilities, and physical forms of such design systems improves and changes rapidly, and therefore understood that the design system illustrated is by way of example and not limitation.

The above description is intended to illustrate preferred embodiments of the present invention. From the discussion above it should also be apparent that the invention can be modified in arrangement and detail by those skilled in the art without departing from the principles of the present invention within the scope of the accompanying claims.